BASEMENT

HD28
.M414
no 884-76

MASS. INST. TECH
APR 21 1977
LIBRARIES

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Alfred P. Sloan School of Management
Center for Information Systems Research

VIRTUAL MACHINE COMMUNICATION FOR THE IMPLEMENTATION OF DECISION SUPPORT SYSTEMS

John J. Donovan
Henry D. Jacoby

REPORT CISR-28
SSWP 884-76

December 1976

Center for Information Systems Research
Massachusetts Institute of Technology
Alfred P. Sloan School of Management
50 Memorial Drive
Cambridge, Massachusetts, 02139
617 253-1000

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Alfred P. Sloan School of Management
Center for Information Systems Research

VIRTUAL MACHINE COMMUNICATION FOR THE IMPLEMENTATION OF DECISION SUPPORT SYSTEMS

John J. Donovan
Henry D. Jacoby

REPORT CISR-28
SSWP 884-76

December 1976

HD28
.M414
no. 971-76

M.I.T. LIBRARIES
APR 2 1 1977
RECEIVED

VIRTUAL MACHINE COMMUNICATION FOR THE IMPLEMENTATION OF DECISION SUPPORT SYSTEMS

ABSTRACT

1. Introduction

2. Architecture of Software

    2.1 Ease of Transport

    2.2 Enhancement of Data Management System

    2.3 Multiple Users Accessing the Same Database

    2.4 Single User Access to Multiple Database Systems

3. Software Engineering Techniques

    3.1 Communication Mechanisms between VM's

        3.1.1 Use of Virtual Punches and Card Readers

        3.1.2 Other Mechanisms

        3.3.3 Communication Mechanisms Used

        3.1.4 Communication between the User VM and the Manager VM

        3.1.5 Communication between the User VM and the Interface VM

        3.1.6 Communication between the Interface VM and a Database VM

    3.2 Summary of the Functions of the Virtual Machine

        3.2.1 Functions of the Manager Virtual Machine

        3.2.2 Functions of the Database Virtual Machine

        3.2.3 Functions of Communications VM

    3.3 Synchronization of Requests to the Database Virtual Machine

4. Performance Evaluation

    4.1 Some Experimental Observations

    4.2 Some Theoretical Observations

5. Conclusion

REFERENCES

0739935

VIRTUAL MACHINE COMMUNICATION FOR THE IMPLEMENTATION OF DECISION SUPPORT SYSTEMS*

John J. Donovan
Henry D. Jacoby

November 24, 1976

Revision according to review suggestions for publication in IEEE Transactions of Software Engineering -- Virtual Machine Communication for the Implementation of Decision Support Systems (March 1976).

---

* This work has been sponsored in part by the New England Regional Commission (Contract No. 1053068). The M.I.T. effort is carried out within the Center for Information Systems Research of the Sloan School of Management, in collaboration with the Energy Laboratory. The work has been greatly assisted by an M.I.T./IBM joint study on information systems. Special recognition is due to the staff of the IBM Cambridge Scientific Center -- especially to Dr. Rafael Fessel for his assistance in implementing these ideas, to Dr. Richard MacKinnon, who is head of that Center, and to Drs. William Timlake and Stuart Greenberg, who have been coordinators of the joint study. We also wish to acknowledge the members of the IBM San Jose Research Center for the use of their relational data-base system, SEQUEL, and for their assistance in adapting that support system for use in the application described here.

ABSTRACT

This paper presents the use of the virtual machine concept as a software engineering tool. The focus is on techniques that allow the rapid integration and assimilation of existing data, models, analytical facilities, report generation facilities, and database management systems. Since many of these models, programs, and facilities run under different operating systems, they often are incompatible with each other. By combining multiple virtual machines in a particular configuration, and allowing for communication among them, it has been possible to overcome these difficulties. The result is a set of software engineering tools that seem particularly useful in decision support systems. We present the application of these tools along with the software techniques used to implement them, and quantify some of their costs and benefits.

1. INTRODUCTION

A "decision support" system is a management information system designed to support decisions being made by a manager or policy-maker [Scott Morton, 1976]. As the complexity of modern society increases, the requiements of computational systems to assist the decision-maker grow correspondingly. For example, which of us would try to solve a fifth-order differential equation by inspection? Yet every day corporate managers, government officials, and other decision-makers are asked to make decisions by inspection, on events which are no less complex.

The computational needs of a decision support system include: a capability to store, validate, update, and access data (a data management capability); a capability to perform computations using those data (an analytical capability, including modeling and statistical facilities); a capability to present the desired information in a concise way (a report generation capability); and a capability to quickly assemble and adjust the programs to meet changing purposes. It is this last computational need that traditional approaches to MIS often handle inadequately, for in many information systems a major change may take months or even years to implement, while the initial construction may take even longer.

It has been pointed out that decision support systems fall into two classes [Donovan and Madnick, 1976]. There are Institutional Decision Support Systems that deal with decisions of a recurring nature (for example, a financial portfolio management system) and Ad-hoc Decision Support Systems that deal with problems that are not usually anticipated or recurring (for example, the decision to support or oppose a nuclear power moratorium). Traditionally, the great bulk of the effort in developing Institutional

Decision Support Systems has been focused on tuning such systems, as they are used over and over again. Nonetheless, when such systems are first brought on-line, they frequently undergo major changes as organizational or human-factor considerations dictate revisions in the database, the reports generated, and the computations that are needed. In many cases, the deficiencies of a system can be determined only after it is in use. Hence, what is needed are software engineering facilities to breadboard such information systems quickly so that users may experiment with them. The tuning process of developing the most efficient system can come later.

The user of an Ad-hoc Decision Support System needs whatever information is available to support his decision. Often the choice is going to be made anyway (should the company merge or not?), and usually on a close deadline. In this case, the speed and costs of developing the information are the dominant criteria. Less focus needs to be placed on the operational costs since such systems are seldom used in an operational mode over a long period of time.

Therefore, in both types of decision support systems a need exists for software facilities for rapid and inexpensive construction. The computational tools must have the ability to integrate and consolidate existing models, programs, and databases. The implementor of the decision support system must be able to use the languages with which he is most familiar, for often there is insufficient time to learn new ones. Users also must be able to access the packages most suited for their application, and thus they must have the ability to use different modelling languages, statistical languages, and analytical techniques in an integrated environment. Further, we have found it useful for multiple users to be able to access the same

database, using the software facilities or analytical packages of their choice. Often, of course, an individual user may want to use a system ordinarily considered to be incompatible with those being used by others, and this circumstance needs to be accommodated as well.

Software engineering tools that can meet these requirements offer several advantages. There is less need to retrain analysts in order for them to gain access to a particular database, and hence, a reduction in time to implement a decision support system. Moreover, a wider use can be made of available software in any system development. Thus a major emphasis of our work has been on reducing the costs and the time required to integrate programs, modelling systems, and programming languages.

2. ARCHITECTURE OF THE SOFTWARE

The software architecture used to achieve this reduction in costs and time makes extensive use of the virtual machine (VM) concept [Parmelee, 1972, Goldberg, 1973; Donovan and Jacoby, 1975]. A virtual machine may be defined as a replica of a real computer simulated by a virtual machine monitor (VMM), a software program, and appropriate hardware support. For example, the VM/370 [IBM, 1] system allows a single IBM system/370 to appear functionally as though it were multiple independent system/370s' (i.e., multiple virtual machines). Thus a VMM can make one computer system function as thought it were multiple, physically isolated systems. Some advantages of virtual machines have been discussed in the literature [Madnick, 1969; Buzen and Gagliardi, 1973; Madnick and Donovan, 1974]. We present further uses of this concept.

The configuration of virtual machines that we have found particularly helpful is depicted in Figure 1. Each box denotes a separate virtual machine. The boxes across the top of the figure represent virtual machines executing different user-oriented programs (modelling, analytical, statistical, editorial, etc.). The boxes along the bottom of the figure denote virtual machines executing different database management systems (systems that can input and retrieve data). The boxes in between denote interface virtual machines that run programs to connect any of the database machines to any of the analytical systems.

A user can access any modelling or database machine, or any combination of modelling machines connected to a database machine, by specifying (through the mechanism of a virtual machine) which machines are to be interconnected.

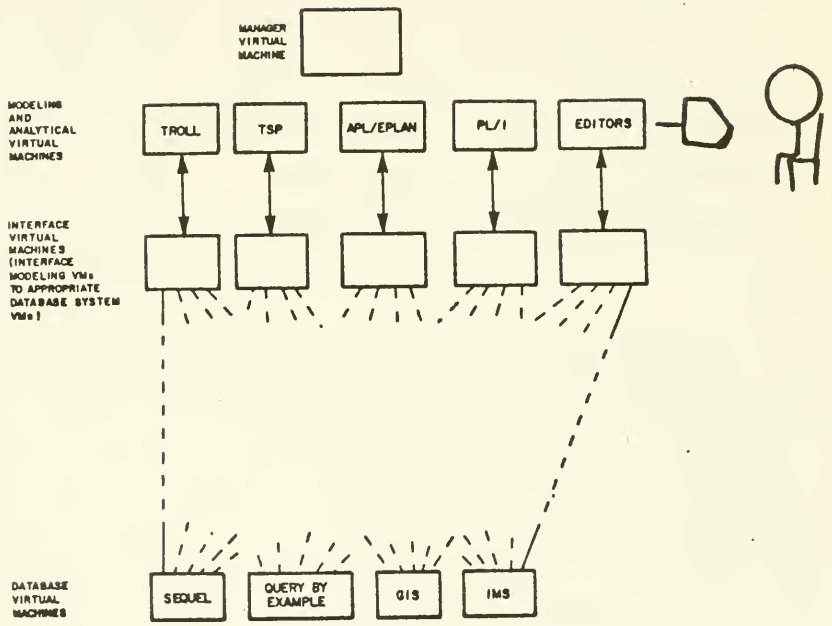

Figure 1. Schema of Virtual Machines

We have called this schema the Generalized Management Information System (GMIS) [Donovan and Jacoby, 1975].[1] Although it has been implemented on an IBM/370, we foresee and advocate such schema being implemented on other manufacturers' machines as other VM systems become available. Further, the approach would appear to be applicable to a network configuration involving different (perhaps remote) physical machines. However, we have not thoroughly investigated this extension to date.

Since each virtual machine can be configured to run any IBM/360 or 370 operating system, any program that runs on the 360 or 370 equipment can be transferred to a virtual machine along with the appropriate operating system.

2.1 The Interconnection of Data Management and Analysis Systems

In breadboarding a system it often proves necessary to access the data in ways not originally thought of. Hence, a flexible data management facility is often necessary. Often such data management faciliites have poor data analysis, statistical or report generation facilities, or none at all. One instance of the schema of virtual machines of Figure 1 would consist of three interconnected machines: a modelling machine connected (via an interface machine) to a database machine. For example, we have found a commonly used configuration is one involving the database language, SEQUEL, and APL. Such a configuration extends the flexible data management capabilities of SEQUEL [Chamberlin, 1974] with the good analytical capabilities of APL [Pakin, 1972] as well as the econometric

---

[1] We acknowledge Louis M. Gutentag for his assistance in supervising the many M.I.T. students who have worked in implementing this system and in coordinating the IBM efforts.

modelling facilities of EPLAN [Schober, 1974] which is imbedded in APL.

Similarly, the GMIS schema can provide for the enhancement of any analytical or statistical capability by simply and quickly adding a database capability underneath it.

Let us take a specific example of how such a set up may appear to the user. The example is taken from a recent decision support application in the New England Energy Management Information System (NEEMIS) Project [NERCOM, 1976] involving analysis of energy consumption within state buildings. The goal of this particular application was to support decisions on how and where to initiate energy consumption programs. Figure 2 depicts a user session where the system user has access to a relational-based data management capability along with the APL/EPLAN analytical package. (Note the interface VM is transparent to the user.) APL/EPLAN has limited data management capability but relatively good analytical properties. The analytical features are extended by adding the flexible database management capabilities of a system like SEQUEL. The objective of the user in the particular session depicted in Figure 2 is to make a monthly plot of the consumption of the Hermann Building and of the Sloan Building for a particular year. Note: not depicted in the session are the user commands to configure the particular schema for Figure 2.

The QUERY commands are invoked form the modelling facility. These commands pass the SEQUEL database SELECT command down to the database system. The database system returns the requested data as a vector (e.g., quantity, Sloan consumption, Hermann consumption, size). The user then invokes appropriate APL functions to normalize the consumption vectors for the differences in square footage between the two buildings. The user

then invokes the plot function of EPLAN to display the desired information.[2]

2.2 Multiple Users Accessing the Same Database

In addition to allowing models and databases to communicate with each other, and facilitating the transport of a program running under any 370 operation system to another 370/360 computer, this configuration presents several other advantages. It allows multiple users working on the same problem to access the same database. An example of such a configuration would consist of three virtual machines connected to a single virtual machine that is running a database system. With such a configuration one user may use TROLL [NBER, 1975]; another may use TSP [Hall, 1975], which is an econometric modelling language; and a third may use FORTRAN. This is possible with all running under different operating systems with all users requesting data stored in the single database management system. An important point to stress here is that TROLL is incompatible with FORTRAN since TROLL must be run under its own operating system, which is a non-standard IBM system.

2.3 Single User Access to Multiple Database Systems

GMIS schema of Figure 1 also allows access and maintenance of data series on several different data management systems. Such an instance of the GMIS schema could consist of a single user accessing a FORTRAN machine

---

[2] In this particular application the discovering of significant discrepancies in heating consumption in state buildings served to motivate further study of the buildings. This subsequent analysis used several existing models of heating consumption in buildings (e.g., NECAP [Henninger, 1975]). These models had been written under several different operating systems, and thus there was a great advantage in being able to simultaneously run these different systems.

which could be connected to two different database machines, e.g., SEQUEL and QBX [Zloof, 1975]. In Ad-hoc decision support systems this is particularly helpful where answers may be needed quickly, and there is often no time to transport the appropriate data series and models to a common system.

3. SOFTWARE ENGINEERING TECHNIQUES USED

The architecture makes heavy use of the virtual machine concept. However, it is important to note that the advantages that have been expressed in the literature (e.g., increased security, reliability, and the ability to simultaneously run different operating systems) are largely a result of isolation of each virtual machine [Donovan and Madnick, 1975]. That is, each simulated machine is autonomous. The original philosophy of the VM concept was isolation -- each virtual machine should be unaware that other VM's exist. Until recently, applications of VM technology were consistent with this philosophy. However, traditional operating system primitives of interprocess communication (e.g., 'P' and 'V" [Dijkstra, 1968]) that are implemented within one operating system are not, in their present implementation, capable of communicating or synchronizing with another operating system executing on another computer (virtual machine).

Essentially, what is needed is a means of passing commands (e.g., database queries) and data to the database machine, a means of returning data, a locking and querying mechanism, and mechanisms for converting data into compatible forms. This section discusses the mechanism used.

3.1 Communication Mechanisms between VM's

3.1.1 Use of Virtual Punches and Card Readers

One mechanism to perform communication between virtual machines (for example, a modelling VM and the database VM) is to use virtual card readers and card punches. In this case, the database machine would be in wait state trying to read a card from its virtual card reader. The

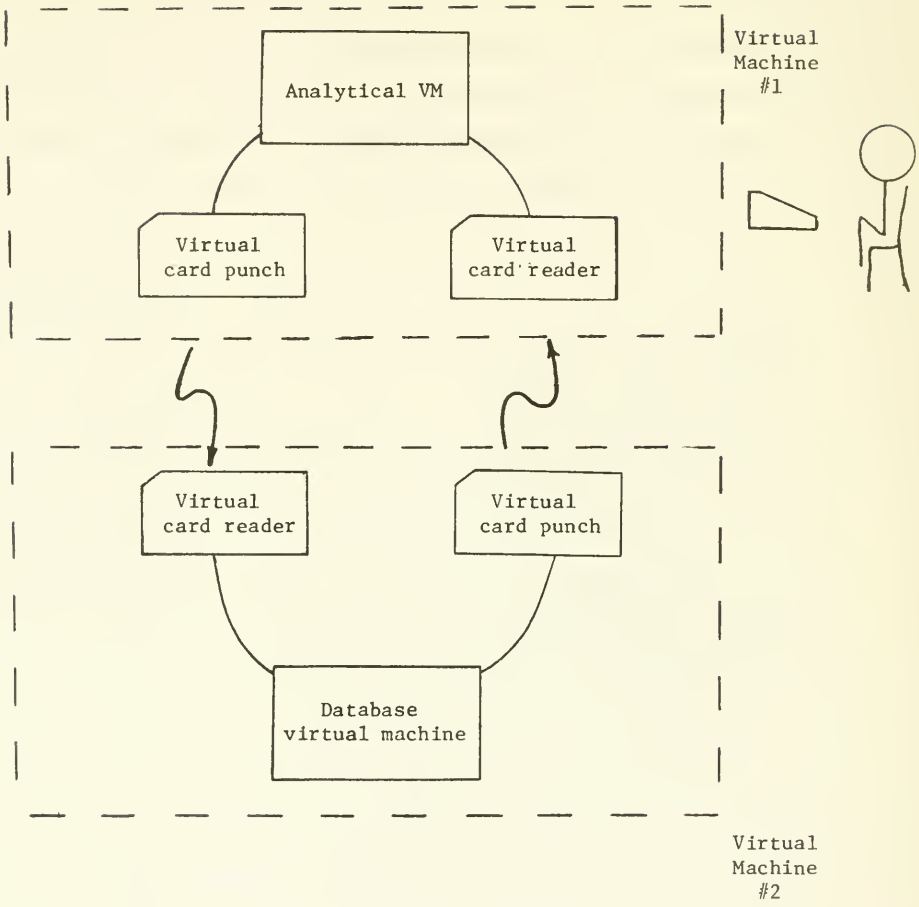

Figure 3. Communication between Two Virtual Machines via Virtual Card Readers and Punches

analytical machine would cause a card to be punched on the virtual card punch. The card would contain the desired command. The card would be read by the database VM (see Figure 3) through the use of a virtual card reader. However, because of the present mechanisms that VM/370 uses to simulate I/O, this mechanism of communication is inefficient and costly.

3.1.2 Other Mechanisms

Other mechanisms that researchers have experimetned with for communications between virtual machines include: the page swap method and the data move method [Hsieh, 1974; Bagley, et al., 1976]; segment sharing [Gray and Watson, 1975] channel-to-channel adaptor and the virtual card punch and reader [Donovan and Jacoby, 1975], which is available with standard releases of VM/370 [IBM,1]. The page swap method has been implemented by IBM using a VM enhancement of the IBM 370 DIAGNOSE INSTRUCTION. The experimental implementation, called SPY, can be thought of as a Core-to-Core transfer between the two communicating virtual machines. This is a very efficient mechanism for communication between virtual machines. However, it requires the receiving VM to be capable of handling an external interrupt. Hence, this mechanism is best used between virtual machines running programs that can call external subroutines, where these external subroutines are, in turn, capable of modifying the interrupt addresses and handling the interrupt.

3.1.3 Communication Mechanisms Used

In the GMIS configuration the more efficient SPY communication mechanism is used whenever convenient. Otherwise shared minidisks are used. Let us discuss which mechanism is used where, and why.

Figure 4 depicts one instance of the general schema of Figure 1, making explicit the communication mechanism used. Two user analytical virtual machines are depicted, one that is executing PL/I Programs, the other executing APL Programs, both connected to a single database machine. In the general schema a user logs into his analytical machine and sends a message to the virtual machine manager, which is being executed in another virtual machine. The message specifies what database virtual machine the user wishes to access. In the particular instance depicted in Figure 4, both users have requested the same database virtual machine, which in that case was a virtual machine running the relational database system SEQUEL.

The virtual machine manager invokes and loads the appropriate routines into the interface virtual machines. These routines perform such functions as the formatting of the data (from the database virtual machine) into the form required by the analytical virtual machine. In the case of APL, for example, the data should be returned in the form of a vector. In the case of PL/I, the data should be returned in the form of a data structure. However, the data as stored in the depicted data management virtual machine are in the form of relations or tables.

The APL user (on the right of Figure 4) must apply the minidisk communication mechanisms, for such a user has difficulty handling external interrupts that the SPY mechanism would require. The user VM for APL may send a transaction to the communications VM by writing it to a CMS file (CMS is a single user operating system commonly run on VM [IBM,2]) and spooling a card from its virtual card punch to the communications VM's virtual card reader that generates an interrupt. The communications VM is alerted to the user's request by the interrupt, reads the transaction

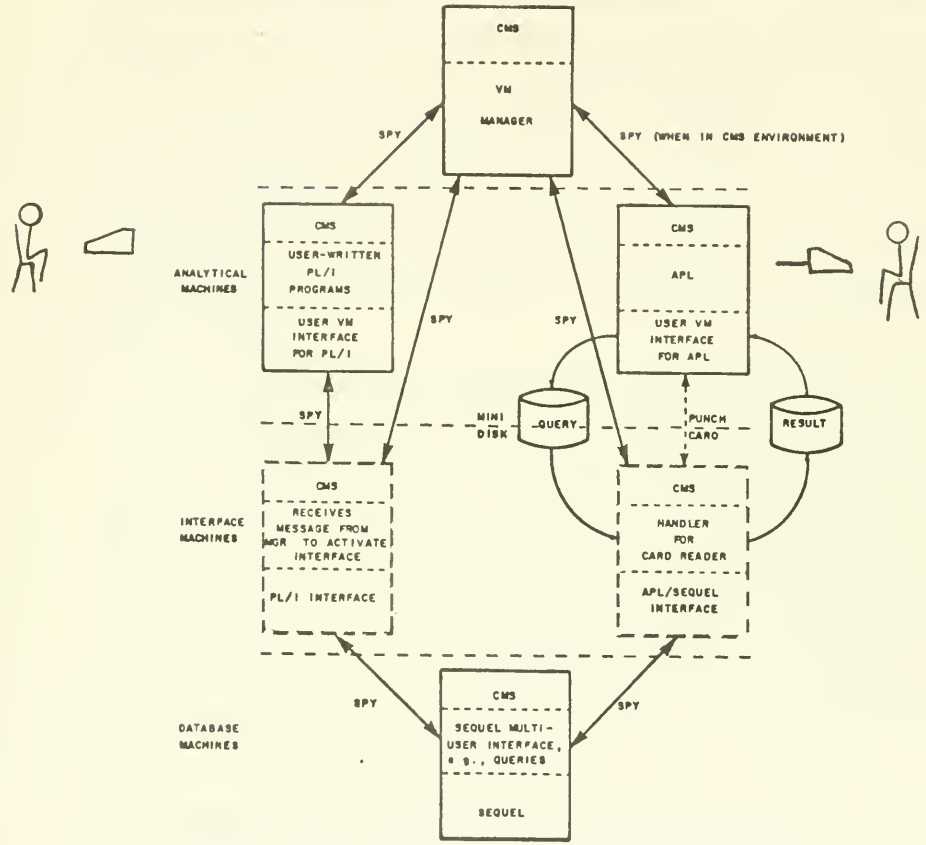

Figure 4. Instance of GMIS with Explicit Communication Mechanisms

from the CMS file, reformats it for the SEQUEL database system, and sends the transaction to SEQUEL VM via the SPY mechanism. After processing the transaction, the SEQUEL VM sends the reply to the communications VM via SPY, the communications VM reformats the reply for APL, writes the reply to a CMS file, and signals the user VM running APL that the transaction is complete by spooling a card to its virtual card reader. The user VM may now read the reply from its CMS file and process it in any manner desired. This entire sequence is illustrated on the righthand side of Figure 4.

Because of the types of programs that are being run in each of the virtual machines depicted in Figure 4, different communication mechanisms were used. For example, since the APL envirnoment does not provide mechanisms for calling subroutines in other languages, it is difficult to incorporate interrupt handlers into an APL user machine. Hence, the SPY mechanism is not used to communicate with an APL machine. All other machines shown in Figure 4 are running programs and operating systems which allow handling of interrupts.

3.1.4 Communication between the User VM and the Manager VM

Since some analytical and modelling software facilities would be difficult to modify for communication directly with the manager VM, a separate communication program (running under CMS) is invoked before the desired facility is activated. This program sends the necessary messages to the manager VM. The user may then activate an analytical or modelling facility under CMS or another operating system. That is, when a user first logs in , he runs under one operating system (CMS) and after

communicating his needs with the manager VM, he loads the desired operating system into his VM.

3.1.5 Communication between the User VM and the Interface VM

For PL/I, TSP, and other software facilities running under CMS, CPY is used to send messages to the interface machine (note that we modified TSP to run under CMS). However, for systems like APL and TROLL that run under their own environments, communication is via minidisks, since standard versions of these systems have the capability of reading/writing disks, as well as punching and reading cards. The message is written on a shared minidisk. The interface VM is notified that such a message is waiting by punching a card on a virtual card reader. The interface VM that has been in wait state reads that card and then reads the message on the minidisk.

3.1.6 Communication between the Interface VM and a Database VM

SPY is used when the database VM is running in a CMS environment (e.g., in the case of SEUQEL and Query by Example). However, communication is via minidisk, virtual card readers, and punches for database systems that do not run in a CMS environment, as would be the case with IMS [IBM,3] in an OS/VS1 environment.

3.2 Summary of the Functions of the Virtual Machine

3.2.1 Functions of the Manager Virtual Machine

The primary function of the manager virtual machine is to respond to user requests to create the connections between the virtual machines by

activating the necessary interface virtual machine (IVM) and database management virtual machine (DBVM). The other function of the manager is to disconnect and automatically log out the appropriate IVM's and DBVM's once the user has finished with them.

To accomplish these functions, several procedures were added to the software running on user VM's and the manager VM. When a user logs into his user VM, he makes a request to connect to a database VM (through his interface VM) by sending a message to the manager VM. The user-initiated action causes the manager VM to receive an external interrupt. The external interrupt handlers that have been added to the manager VM perform the following: (a) check ID of sender for authorization; (b) look at the message sent by the sending VM. If the message is to log in an IVM, then it will check to see if such a VM is already running. If not, it automatically logs one in (note that the manager VM has operator privileges that permit it to log in other virtual machines). The manager VM then sends a message to the CVM for it to load the appropriate interface module. The manager VM then sends a completion code to the user VM. If the completion code message were a terminate message, the manager would automatically log off the user's IVM. Furthermore, the manager periodically checks all IVM's to see if they have an owner, i.e., whether or not the user VM's are currently logged in. If a communications VM does not have an owner, the manager VM automatically logs off the user's CVM.

3.2.2 Functions of the Database Virtual Machines

The database virtual machine executes programs concerned with storing and accessing data as well as storing the data itself. Any database

system may run in such a machine. Presently, GMIS provides users with access to an interactive relational database management system called SEQUEL, which was developed at the IBM San Jose Research Laboratory. We are presently adding the Query by Example relational database system which is being developed at the IBM Yorktown Heights Research Laboratory [Zloof, 1975]. These relational systems allow database transactions to be entered on-line, and prepare replies to these transactions in the form of single-valued results or tabular reports.

A database VM, regardless of the database management system running on it, has additional software that receives transactions from the interface virtual machines belonging to different users and stacks these requests in the order in which they are received. Each request is processed (one at a time) by the database management facility, and the reply is passed back to the interface VM that sent the transaction request. After each reply is sent, the database machine selects the next request from the stack, identifies the sending interface VM, and processes the transaction. This processing scheme provides a multiple-user environment for each database VM. Also, GMIS supports multiple database VM's, each processing transactions against a different physical database, as shown in Figure 1.

3.2.3 Functions of Interface Virtual Machine

The interface VM's provide mechanisms for user VM's to interface with database VM's. When a user VM signals the manager VM to activate a configuration of VM's, this user VM indicates in which modelling or analytical environment it is currently running, and to which database VM it wished to

send transactions. The manager VM uses this information to signal an interface VM to load the appropriate interface routines for the particular user environment/database system combination desired.

Each interface routine is custom built to permit communications between a specific user environment and a specific database system. Any reformatting of transactions from the user or replies from the database system is handled by the interface routine that resides in the user's communications VM.

3.3 Synchronization of Requests to the Database Virtual Machine

The synchronization of transactions (access, write) from a multi-user configuration (e.g., several analytical machines connected to a single database machine) is implemented in the present GMIS configuration as follows: each user interface virtual machine, which is accessed by logging into a separate account under VM/370 (the machine across the top of Figure 1) sends transactions to the database virtual machine through the appropriate communications facility (either SPY or the spooling facility). The multi-user interface (MUI) stacks transaction requests and processes them one-at-a-time using a FIFO basis. While the database machine is processing any transaction, it is locked and all other transactions are queued. The result of each transaction is passed back to the interface VM that made the request. The replies to the transactions are then converted into the appropriate formats by the communication VM. The communication VM passes the requested data to the user interface machine where it is processed as programmed by the user.

4. PERFORMANCE EVALUATION

We have begun to evaluate both collectively and individually the performance costs of the technologies used in the GMIS schema. The conclusion of the initial work is that measurable degradations in computer time occur in executing software in a GMIS environment. However, these costs are more than compensated for, in certain applications, by the decrease in fixed costs of developing such applications.

To place thse costs into a framework relative to traditional technologies, we refer to Figure 5 where fixed costs (costs of developing a management information system) and variable costs (costs of operating such systems) are depicted for these systems. The units of the y axis are dollars; the units of the x axis are time or number of queries made.

The dashed line represents costs typically found in traditional management information systems. For example, in a payroll system the focus is on low variable costs (slope of line is small) as each check issued must cost only pennies.

The 0 curve represents typical costs associated with the development and operation of an institutional decision support system (DSS) using traditional technologies. For example, a portfolio management system often is first brought into operation for a short time only to find additional fixed costs must be incurred due to changes in perception of the function of the system or in available data. Typically, these changes become less frequent until finally stabilizing, and attention is then given to tuning the system (reducing the slope of the curve).

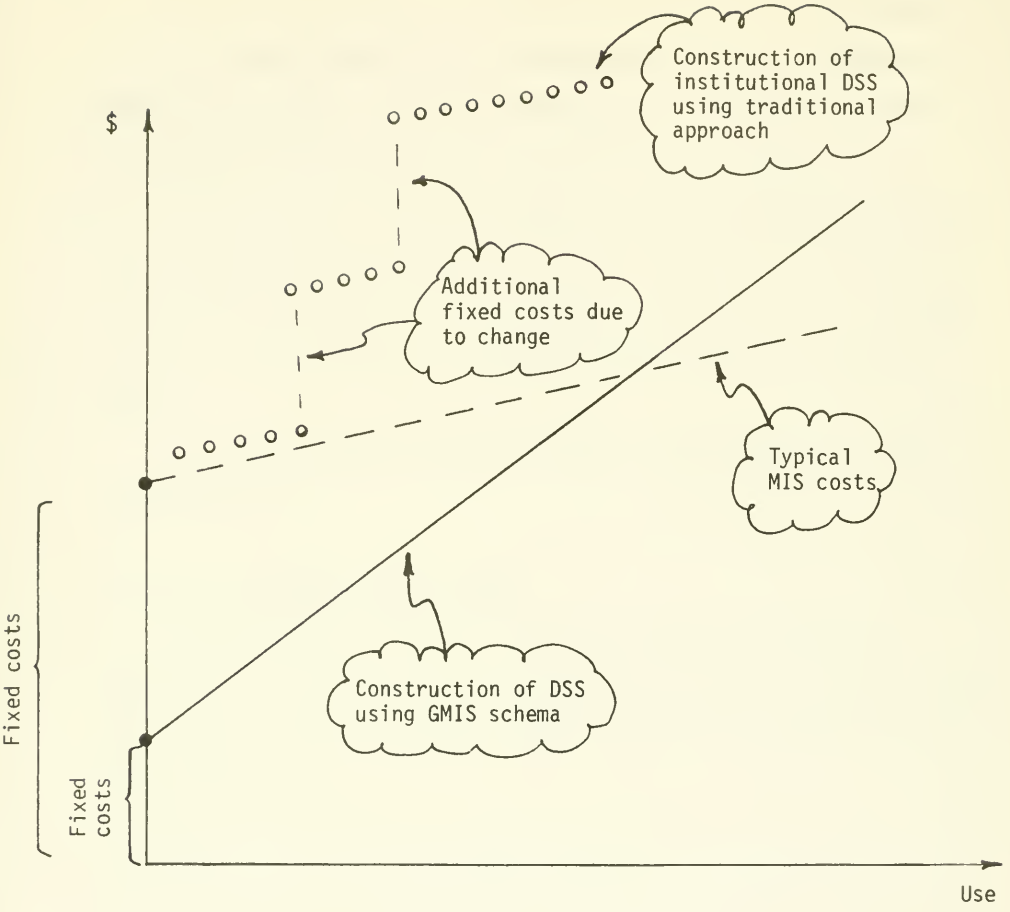

Figure 5. Fixed and Variable Costs of Magagement Information Systems

The solid line depicts a more desirable cost curve for DSS (both ad-hoc and during breadboarding, institutional) as they are seldom in operation long enough for the break-even point (intersection with the dashed line) to be reached.

Our experience is that the technologies used in GMIS-type configurations do in fact allow for the lower fixed costs depicted in the solid curve, and that the total cost is less even taking account of the higher variable costs as depicted in the larger slope of the solid curve of Figure 5.

This section presents some of our preliminary findings on these variable costs.

4.1 Some Experimental Observations

By experimental observation we find that the SPY communication mechanism is approximately three times faster in processing average queries (e.g., those in Figure 3) than the shared minidisk commnication mechanism described earlier.

We have experimentally confirmed the intuitive result that the fraction of the total time spent in the communication mechanism is inversely proportional to the complexity and amount of data requested. That is, the time to process complex queries is mostly spent in the data management machine. Hence, the overhead associated with the communication mechanism is less important. For the types of queries in Figure 3, the time spent in the communication mechanism is under 8% of the total time to process those queries. This time could be further reduced with hardware to assist in such communication (e.g., P and V hardware operations [Dijkstra, 1968] between VM's).

For most of the models that have been transferred onto the system, we have observed a degradation of performance of less than 10% over what these models would run if they were running on a real machine under their own operating system. The VM configuration we are presently using is an IBM 370, 158 located at the IBM Cambridge Scientific Center.

4.2 Some Theoretical Observations

One of the schema commonly used by users of the GMIS system is depicted in Figure 6 where multiple modelling machines are connected to a single database machine. (Once again, the interface machines are transparent to the user and hence have been omitted). The particular synchronization method used to handle the requests from each of the modelling machines is to lock-out all modelling machines while a request is being processed by the database machine. Hence, user response of a locked-out machine is degraded. What is the degradation of performance with each additional user? What is the best locking strategy? What are the implications of this architecture of VM on VM scheduling algorithms to improve performance?

An access on an update on a database machine may be initiated either by a user query, which would then be passed on the interface machine to the modelling machine to the database machine, or by a model executing on the modelling machine. In either case, under present locking strategy, the database machine, while processing the request, locks-out (q's) all other requests. An analysis of this potential degradation is complicated by the fact that as some VM's become locked, others get more of the real CPU's time, and therefore generate requests at a faster rate with a limit reached when the entire CPU is devoted to one modelling machine and one

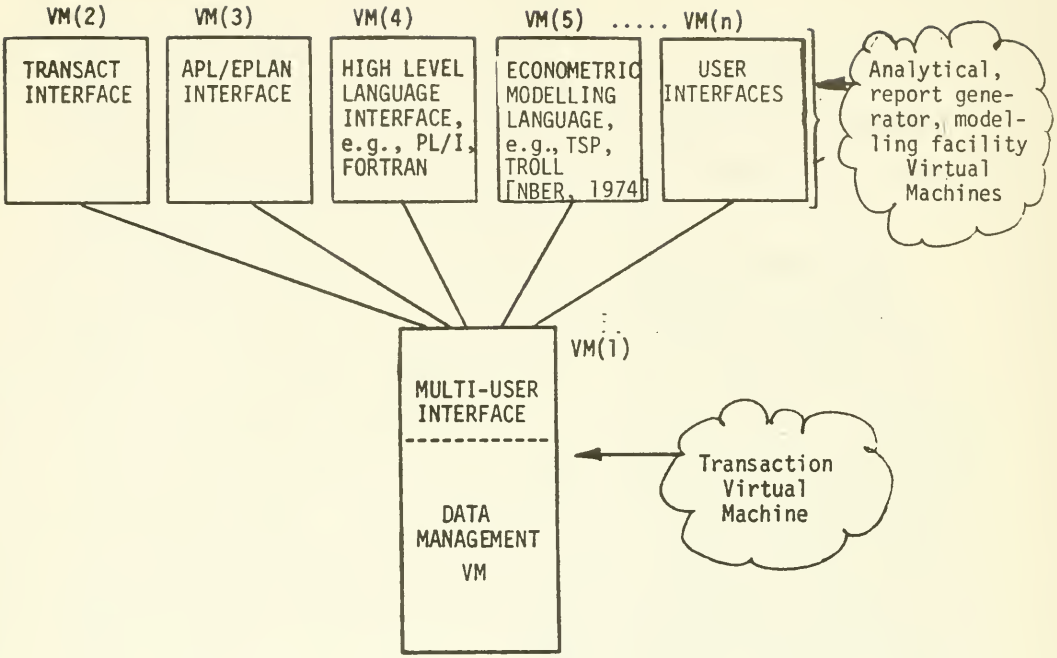

Figure 6. Multiuser GMIS Configuration

database machine. However, the database machine also gets more of the CPU's time and therefore processes requests faster. For example, if there are ten virtual machines, each one receives one tenth of the real CPU. However, if seven of the ten are in lock state, then the remaining three receive one third of the CPU. (Thus these run faster in real time than they did when ten were running).

A Markov model was constructed of this phenomenon,[3] where lamda denotes the request rate, the rate at which the modelling machine made requests of the database management machine, and mu denotes the rate at which the database management machine serviced those requests. The Markov model was used to compute the total amount of time that it took to execute a particular model -- that is, the time that it took to send the request to the database machine, process those requests, return the data to the modelling machine, and continue the execution of the model. Figure 7 depicts results of the analysis. The general shape of the curve shown in the Figure was verified experimentally.

Figure 7 depicts the total amount of time to execute three different models as a function of the number of modelling virtual machines. Let us consider briefly some of the implications of the analysis depicted in Figure 7. Note: for model A, the degradation and performance is minimal. That is, if model A were running with no other modelling machines, it takes a certain number of minutes to execute; if it were running with ten other modelling machines, it takes nearly the same amount of time to execute. That is, very little time is lost in the synchronization

---

[3] We wish to thank Professors Peter Chen and Stuart E. Madnick for assisting in the analytical formulation and solution of the performance equations, and Jeff Buzen for his review of the analysis.

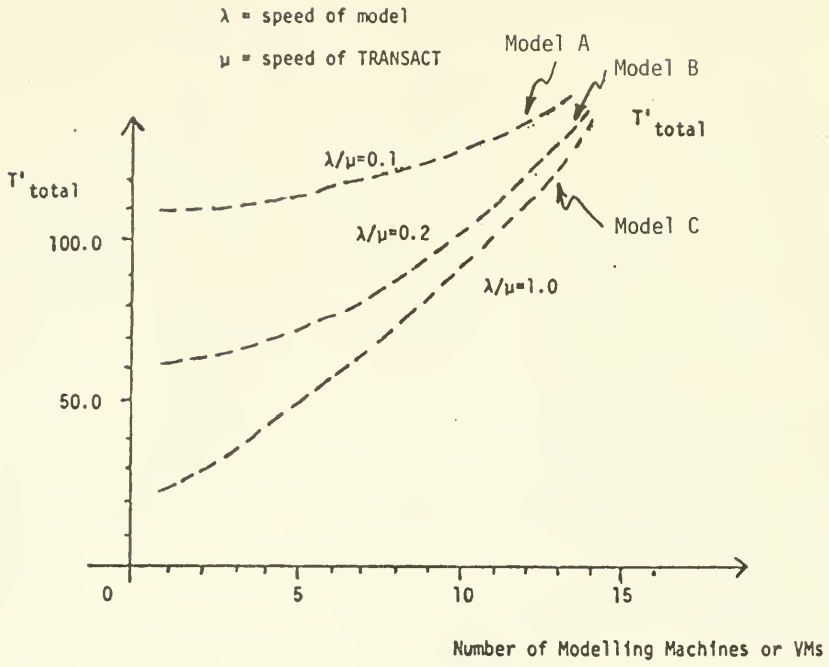

Figure 7. Total Elapsed Times for a VM Configuration

mechanisms. With the ratio so indicated of request rate to service rate, the database management machine is capable of keeping ahead of the request from all the modelling machines. However, model C has a much sharper degradation of performance.

If such a degradation of performance is not tolerable, there are several ways to improve it. The theoretical study would indicate that increasing mu for a given configuration helps performance. Possibly this could be done by changing the processor scheduling algorithm of VM so that the real processor were assigned to the database management system more often, thus speeding it up and increasing mu.

Another way of improving performance loss due to the syncrhonization mechanism would be to change the single locking mechanism used to a multi-locking mechanism. That is, the database management system could lock individual tables of files when they are accessed by a particular modelling machine, allowing other modelling machines that would access other files or tables to also have their requests processed.

Further, the locking strategy could be more selective in that it could only lock an insert command and not lock any portion of the database on a read command. Thus requests could be processed simultaneously for unwritten reads into the table and for reads to different tables. Hence, adding another real processor to the multiple lock VM schema could improve performance.

Using a configuration where multiple database machines each have the same database system and data could improve degradation of response time. In such a multiple database schema, all read reqeusts would operate without a lock. Shared locks between machines would be used to keep all database machines locked until a write request was completely processed.

5. CONCLUSION

We have found great advantage in configuring virtual machines in such a way as to allow machines executing user-oriented software (analytical, report generation, modelling) to communicate with machines executing data management software facilities. It makes possible the integration and transport of programs executing under different operating systems, the enhancement of data management systems, the multiple use of a single database from different analytical facilities, and the integration of different databases. We have used two principal software mechanisms for communication between virtual machines: core-to-core (SPY) and use of shared minidisks. The core-to-core mechanisms appear to be faster but require that users of the two communicating machines be able to write interrupt handlers.

For the application areas we have addressed (decision support systems), the use of this configuration has greatly reduced the fixed costs of implementing such systems. However, we have both experimentally and theoretically observed that the variable costs (operational use of these systems) associated with this schema are higher than traditional approaches. In decision support systems that are operational only for a short time, however, the performance loss is more than compensated by the reduction in fixed costs and the increase in speed of implementation.

REFERENCES

Bagley, J. D., Floto, E. R., Hsieh, S. C., and V. Watson: "Sharing Data and Services in a Virtual Machine System," Proceedings of the ACM SIGOPS Fifth Symposium on Operating Systems Principles, pp. 82-88, November 1975.

Brooks, Lynn A. (Commissioner): "Final Report: Crisis Energy Management Program," State of Connecticut, Department of Planning and Energy Policy, Hartford, Connecticut, September 16, 1976.

Buzen, J. P. and U. O. Gagliardi: "The Evolution of Virtual Machine Architecture," Proceedings AFIPS National Computer Conference, pp. 291-299, 1973.

Chamberlin, D. D. and R. F. Boyce: "SEQUEL" A Structured English Query Language," Proceedings of the ACM SIGFIDET Workshop, Ann Arbor, Michigan, pp. 249-264, May 1974.

Codd, E. F.: "A Relational Model of Data for Large Shared Data Banks," Communications of the ACM, 13, 6, pp. 377-387, June 1970.

Date, D. J.: An Introduction to Database Systems, Addison-Wesley, Reading, Massachusetts, 1975.

Dijkstra, E. W.: "The Structure of the T.H.E. Multiporgramming System," Communications of the ACM, 11, 5, pp. 341-346, May 1968.

Donovan, J. J. and H. D. Jacoby: "GMIS: An Experimental System for Data Management and Analysis," Working Paper No. MIT-EL-75-011WP, M.I.T. Energy Laboratory, M.I.T., Cambridge, Massachusetts, September 1975.

Donovan, J. J. and W. R. Keating: "NEEMIS: Test of Governors Presentation," Working Paper No. MIT-EL-76-002WP, M.I.T. Energy Laboratory, M.I.T., Cambridge, Massachusetts, February 1976.

Donovan, J. J. and S. E. Madnick: "Hierarchical Approach to Computer System Integrity," IBM Systems Journal, 14, 2, pp. 188-202, 1975.

Donovan, J. J. and S. E. Madnick: "Institutional and Ad-hoc Decision Support Systems and Their Effective Use," Working Paper, Center for Information Systems Research, M.I.T., Cambridge, Massachusetts, November 1976.

Donovan, J. J. and S. E. Madnick: "Virtual Machine Advantages in Security, Integrity, and Decision Support Systems," IBM Systems Journal, 15, 3, pp. 270-278, 1976.

Goldberg, R. P.: "Survey of Virtual Machine Research," Computer, 7, 6, pp. 34-35, June 1974.

Gray, J. M. and V. Watson: "A Shared Segment and Inter-process Communication Facility for VM/370," Research Report RJ 1579, IBM Research Laboratory, San Jose, California, February 1975.

Hall, R. E.: TSP Manual, Harvard Technical Report No. 12, Harvard Institute of Economic Research, Cambridge, Massachusetts, April 1975.

Henninger, R.H. (editor): NECAP - NASA's Energy Cost Analysis Program, Part I, User's Manual, and Part II, Engineering Manual, NASA Report No. CR-2590, National Technical Information Service, Springfield, Virginia, September 1975.

Hsieh, S. C.: "Inter-Virtual Machine Communication under VM/370," Form No. GC20-1800, IBM, White Plains, New York, 1976.

IBM 1: "IBM Virtual Machine Facility/370: Introduction," Form No. GC20-1800, IBM, White Plains, New York, 1976.

IBM 2: "IBM Virtual Machine Facility/370: CMS User's Guide," Form No. GC20-1819, IBM White Plains, New York, 1976.

IBM 3: "APL Econometric Planning Language," Form No. SH20-1620 (Product No. 5796DW), IBM, White Plains, New York, 1976.

Madnick, S. E.: "Time-sharing Systems: Virtual Machine Concept vs. Conventional Approach," Modern Data, 2, 3, pp. 34-36, March 1969.

Madnick, S. E. and J. J. Donovan: Operating Systems, McGraw-Hill, New York, 1974.

National Bureau of Economic Research: TROLL Reference Manual, 575 Technology Square, Cambridge, Massachusetts, 1974.

New England Regional Commission, "The New England Energy Management Information System," Boston, Massachusetts, September 1976.

Pakin, S.: APL/360 Reference Manual, Science Research Associates, Chicago, Illinois, 1972.

Schober, F.: "EPLAN: An APL-Based Language for Econometric Modelling and Forecasting," IBM Scientific Center, Philadelphia, Pennsylvania, 1974.

Scott Morton, M. S.: "Management Decision Systems: Computer Based Support for Decision Making," Division of Research, Graduate School of Business Administration, Harvard University, Boston, Massachusetts, 1972.

Zloof, M. M.,"Query by Example," Proceedings AFIPS 1975 National Computer Conference, Vol. 44, AFIPS Press, Montvale, New Jersey, pp. 431-437.

BASEMENT

Date Due

NOV 17 '78
FEB 15 '79
MAY 16 1985
AUG 19 1980
OCT 14 '80
MAY 20 1991
OCT. 25 1992
MAY 07 '81
OCT 22 '81

HD28.M414 no.878-76
Donovan, John /A note on performance o
729234 D*BKS 00028912
3 9080 000 761 939

T-J5 143 w no.879-76
Tichy, Noel M./Applied behavioral scie
729233 D*BKS 00028674
3 9080 000 759 024

T-J5 143 w no.880- 76
Zannetos, Zeno/Economic theory and oce
730418 D*BKS 00032844
3 9080 000 809 423

HD28.M414 no.881- 76
Merton, Robert/Continuous-time portfol
730420 D*BKS 00032843
3 9080 000 809 399

T-J5 143 w no.882- 76
Lilien, Gary L/Emerging approaches to
731008 D*BKS 00035424
3 9080 000 840 345

HD28.M414 no.883- 76
Donovan, John /Institutional and ad-ho
730044 D*BKS 00031737
3 9080 000 795 838

HD28.M414 no.884- 76
Donovan, John /Virtual machine communi
730935 D*BKS 00035435
3 9080 000 840 667